MDPI

Article

A Clustering System for Dynamic Data Streams Based on Metaheuristic Optimisation

Jia Ming Yeoh[1], Fabio Caraffini[1]*, Elmina Homapour[1], Valentino Santucci [2] and Alfredo Milani [3]

[1] Institute of Artificial Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester (UK); jiamingyeoh@gmail.com; fabio.caraffini@dmu.ac.uk; elmina.homapour@dmu.ac.uk
[2] University for Foreigners of Perugia, Perugia (Italy); valentino.santucci@unistrapg.it
[3] University of Perugia, Perugia (Italy); alfredo.milani@unipg.it
* Correspondence: fabio.caraffini@dmu.ac.uk

Version December 10, 2019 submitted to Mathematics

Abstract: This article presents OpStream, a novel approach to cluster dynamic data streams. The proposed system displays desirable features, such as a low number of parameters, good scalability capabilities to both high-dimensional data and numbers of clusters in the data set, and it is based on a hybrid structure using deterministic clustering methods and stochastic optimisation approaches to optimally centre the clusters. Similarly to other state-of-the-art methods available in the literature, it uses "microclusters" and other established techniques, such as density-based clustering. Unlike other methods, it makes use of metaheuristic optimisation to maximise performances during the initialisation phase, which precedes the classic online phase. Experimental results show that OpStream outperforms the state-of-the-art in several cases and it is always competitive against other comparison algorithms regardless of the chosen optimisation method. Three variants of OpStream, each coming with a different optimisation algorithm, are presented in this study. A thorough sensitive analysis is performed by using the best variant to point out OpStream robustness to noise and resiliency to parameters changes.

Keywords: dynamic stream clustering; online clustering; metaheuristics; optimisation; population-based algorithms; density-based clustering; k-means centroid; concept-drift; concept-evolution

1. Introduction

Clustering is the process of grouping homogeneous objects based on the correlation among similar attributes. This is useful in several common applications that require the discovery of hidden patterns among the collective data to assist decision making, e.g. bank transaction fraud detection [1], market trend prediction [2,3], and network intrusion detection system [4]. Most traditional clustering algorithms developed relies on multiple iterations of evaluation on a fixed set of data to generate the clusters. However, in practical applications, these detection systems are operating daily, whereby millions of input data points are continuously streamed indefinitely, hence imposing speed and memory constraints. In such dynamic data stream environments, keeping track of every historical data would be highly memory expensive and, even if possible, would not solve the problem of analysing big data within the real-time requirements. Hence, a method of analysing and storing the essential information of the historical data in a single pass is mandatory for clustering data streams.

In addition, dynamic data clustering algorithm needs to address two special characteristics that often occurs in data streams which is known as *concept-drift* and *concept-evolution* [5]. Concept drift refers to the change of underlying concepts in the stream as time progress, i.e. the change in the

relationship between the attributes of the object within the individual clusters. For example, customer behaviour in purchasing trending products always changes in between seasonal sales. Meanwhile, concept evolution occurs when a new class definition has evolved in the data streams, i.e. the number of clusters has changed due to the creation of new clusters or deprecation of old clusters. This phenomenon often occurs in the detection system whereby an anomaly has emerged in the data traffic. An ideal data stream clustering algorithm should address these two main considerations to effectively detect and adapt to changes in the dynamic data environment.

Based on recent literature, metaheuristics for black-box optimisation have been greatly adopted in traditional static data clustering [6]. These algorithms have a general-purpose application domain and often displays self-adaptive capabilities, thus being able to tackle the problem at hand, regardless of its nature and formulation, and return near-optimal solutions. For clustering purposes, the so-called "population-based" metaheuristic algorithms have been discovered to be able to achieve better global optimisation results than their "single-solution" counterparts [7]. Amongst the most commonly used optimisation paradigms of this kind, it is worth mentioning the established Differential Evolution (DE) framework [8–10], as well as more recent nature-inspired algorithms from the Swarm Intelligence (SI) field, such as the Whale Optimisation Algorithm (WOA) [11] and the Bat-inspired algorithm in [12], here referred to as BAT. Although the literature is replete with examples of data clustering strategies based on DE, WOA and BAT for the static domain, as e.g. those presented in [13–16], a little is done for the dynamic environment due to the difficulties in handling data streams. The current state of dynamic clustering is therefore unsatisfactory as it mainly relies on algorithms based on techniques, such as density microclustering and density grid-based clustering, which requires the tuning of several parameters to work effectively [17].

This paper presents a methodology for integrating metaheuristic optimisation into data stream clustering, thus maximising the performance of the classification process. The proposed model does not require specifically tailored optimisation algorithms to function, but it is a rather general framework to use when highly dynamics streams of data have to be clustered. Unlike similar methods, we do not optimise parameters of a clustering algorithm but use metaheuristic optimisation in its initialisation phase, in which the first clusters are created, by finding the optimal position of their centroids. This is a key step as the grouped points are subsequently processed with the method in [18] to form compact, but informative, *microclusters*. Hence, by creating the optimal initial environment for the clustering method, we make sure that the dynamic nature of the problem will not deteriorate its performances. It must be noted that microclusters are lighter representations of the original scenario which are stored to preserve the "memory" of the past classifications. These play a major role since aid subsequent clustering processes when new data streams are received. Thus, by a non-optimal microclusters store in memory can have catastrophic consequences in terms of classification results. In this light, our original use of the metaheuristic algorithm finds its purpose and results confirm the validity of our idea. thee proposed clustering scheme efficiently track changes and spot patterns accordingly.

The remainder of this paper has the following structure:

- section 2 discusses the recent literature and briefly explains the logic behind the leading data stream clustering algorithms;
- section 3 establishes the motivations and objectives of this research and presents the used Metaheuristic Optimisation methods, the employed performance metrics, and the considered data sets for producing numerical results;
- section 4 gives a detailed description of each step involved in the proposed clustering system, clarifies its working mechanism and show methodologies for its implementation;
- section 5 describes the performance metrics used to evaluate the system and provides experimental details to reproduce the presented results;
- section 6 presents and comments on the produced results, including comparison among different variants of the proposed system, over several evaluation metrics;
- section 7 outlines a thorough analysis of the impact of the parameter setting for the optimisation algorithm on the overall performance of the clustering system;

- section 8 summarises the conclusions of this piece of research.

2. Background

There are two fundamentals aspects to take into consideration in data stream clustering, namely *concept-drift* and *concept-evolution*. The first aspect refers to the phenomenon when the data in the stream undergoes changes in the statistic properties of the clusters with respect to the time [19,20] while the second to the event when there is an unseen novel cluster appearing in the stream [5,21].

Time window models are deployed to handle concept-drift in data streams. These are usually embedded into clustering algorithms to control the quantity of historical information used in analysing dynamic patterns. Currently, there are four predominant window models in the literature [22]:

- the "damped time window" model, where historical data weights are dynamically adjusted by fixing a rate of decay according to the number of observations assigned to it [23];
- the "sliding time window" model, where only the most recent past data observations are considered with a simple First-In-First-Out (FIFO) mechanism as in [24];
- the "landmark time window" model, where the data stream is analysed in batches by accumulating data in a fixed-width buffer before being processed;
- the "tilted time window" model, where granularity level of weights gradually decreases as data point gets older.

As for concept-evolution, most of the existing data stream clustering algorithms are designed following a two-phases approach, i.e. consisting of an *online* clustering process followed by an *offline* one, which was first proposed in [25]. In this work, the concept of microclusters was also defined to design the so-called "CluStream" algorithm. This method forms microclusters having statistical features representing the data stream online. Similar microclusters are then merged into macro-clusters, keeping only information related to the centre of the densest region. This is performed offline, upon user request, as it comes with information losses since merged clusters can no longer be split again to obtain the original ones.

In terms of online microclustersing, most algorithms in the literature are distance-based [22,26,27], whereby new observations are either merged to existing microclusters or form new microclusters based on a distance threshold. The earliest form of distance-based clustering strategy is the process of extracting information about a cluster into the form of a Clustering Feature (CF) vector. Each CF usually consists of three main components: 1) a linear combination of the data points referred to as Linear Sum vector $\overrightarrow{LS}$; 2) a vector $\overrightarrow{SS}$ whose components are the Squared Sums of the corresponding data points components; 3) the number N of points in a cluster.

As an instance, the popular CluStream algorithm in [25] makes use of CF and the tilted time window model. During the initialisation phase, data points are accumulated to a certain amount before being converted into some microclusters. On the arrival of new streams, new data are merged with the closest microclusters if their distance from the centre of the data point to the centre of the microclusters is within a given radius (i.e. ϵ-neighbourhood method). If there is no suitable microclusters within this range, a new microclusters is formed. When requested, the CluStream uses the k-means algorithm [28] to generate macro-clusters from microclusters in its offline phase. It also implements an ageing mechanism based on timestamps to remove outdated clusters from its online components.

Another state-of-the-art algorithm, i.e. DenStream, is proposed in [18] as an extension of CluStream using the damped time window and a novel clustering strategy named "time-faded CF". DenStream separates the microclusters into two categories: the potential core microclusters (referred to as *p-microclusters*) and the outlier microclusters (referred to as *o-microclusters*). Each entry of the CF is subject to a decay function that gradually reduces the weight of each microclusters at a regular evaluation interval period. When the weight falls below a threshold value, the affected *p-microclusters* are degraded to the *o-microclusters*, and they are removed from the *o-microclusters* if the weights deteriorates further. On the other hand, *o-microclusters* that have their weights improved

are promoted to p-microclusters. This concept allows new and old clusters to gradually form online, so addressing the concept evolution issue. In the offline phase, only the *p-microclusters* are used for generating the final clusters. Similar p-microclusters are merged employing a density-based approach based on the ϵ-neighbourhood method. Unlike other commonly used methods, in this case clusters can assume an arbitrary shape and no a-priori information is needed to fox the number of clusters.

An alternative approach is given in [29], where the proposed STREAM algorithm does not store CF vectors but directly compute centroids on-the-flight. This is done by solving the "k-Median clustering" problem to identify the centroids of K clusters. The problem is structured in a form whereby the distance from data points to its closest cluster has associated costs. Using this framework, the clustering task is defined as a minimisation problem to find the number and position of centroids that yield the lowest costs. To process indefinite length of streaming data, landmark time window is used to divide the streams into n batches of data, and the K-median problem solving is performed on each chunk. Although the solution is plausible, the algorithm is evaluated to be time-consuming and memory expensive in processing streaming data.

The OLINDDA method proposed in [30] extends the previously described centroid approach by integrating the ϵ-neighbourhood concept. This is used to detect drifting and new clusters in the data stream, with the assumption that drift changes occur within the existing cluster region whilst new clusters form outside the existing cluster region. The downside of the centroid approach is that the number of K centroids needs to be known a-priori, which is problematic in a dynamic data environment.

There is one shortcoming for the two-phases approach, i.e. the ability to track changes in the behaviour of the clusters is linearly proportional to the frequency of requests for the offline component [31]. In other words, the higher the sensitivity to changes, the higher the computational cost. To mitigate these issues, an alternative approach has been explored by researchers to merge these two phases into a single online phase. FlockStream [32] deploys data points into a virtual mapping of a two-dimensional grid, where each point is represented as an agent. Each agent navigates around the virtual space according to a model mimicking the behaviour of flocking birds, as done in the most popular SI algorithms, e.g. those in [33–35]. The agent behaviour is designed in a way such that similar (according to a given metric) birds will move in the same direction as its closest neighbours, forming different groups of the flock. These groups can be seen as clusters, thus eliminating the need for a subsequent offline phase.

MDSC [36] is another single-phase method exploiting the SI paradigm inspired by the density-based approached introduced in DenStream. In this method, the Ant Colony Optimisation (ACO) algorithm [37] is used to optimally group similar microclusters during the online phase. In MDSC, a customised ϵ-neighbourhood value is assigned to each cluster to enable "multi-density" clusters to be discovered.

Finally, it is worth mentioning the ISDI algorithm in [38], which is equipped with a windowing routine to analyse and stream data from multiple sources, a timing alignment method and a deduplication algorithm. This algorithm is designed to deal with data streams coming from different sources in the Internet of Things (IoT) systems and can transform multiple data streams, having different attributes, into cleaner data sets suitable for clustering. Thus, it represents a powerful tool allowing for the use of streams classifiers, as a.g. the one proposed in this study, in IoT environments.

3. Motivations, Objectives and Methods

Clustering data streams is still an open problem with room of improvement [39]. Increasing the classification efficiency in this dynamic environment has a great potential in several application fields, from intrusion detection [40] to abnormalities detection in patients physiological streams data [41]. In this light, the proposed methodology draws its inspiration from key features of the successful methods listed in section 2, with the final goal of improving upon the current state-of-the-art.

A hybrid algorithm is then designed by employing, along with standard methods as e.g. CF vectors and the landmark time windows model, modern heuristic optimisation algorithms. Unlike similar approaches available in the literature [37,42,43], the optimisation algorithm is here used during the online phase to create optimal conditions to the offline phase. This novel approach is described in details in section 4.

To select the most appropriate optimisation paradigm, three widely used algorithms, i.e. WOA, BAT and DE, were selected from the literature and compared between them. We want to clarify that the choice of using three metaheuristic methods, rather than other exact or iterative techniques, was made to be able to deal with ch alleging characteristics of the optimisation problem at hand, e.g. the dimensional of the problem can vary according to the data set, the objective functions is highly non linear and not differentiable, which make them not applicable or time-inefficient.

A brief introduction of the three selected algorithms is given below in section 3.1. Regardless of the specific population-based algorithm used for performing the optimisation step, each candidate solution must be encoded as an n-dimensional real-valued vector representing the K cluster centres for initialising the following density-based clustering method.

Two state-of-the-art deterministic data stream clustering algorithms, namely DenStream and CluStream, are also included in the comparative analysis to further validate the effectiveness of the proposed framework.

The evaluation methodology employed in this work consists in running classification experiments over the data sets in section 3.2 and measuring the obtained performances through the metrics defined in section 3.3.

3.1. Metaheuristic Optimisation methods

This section gives details on the implementation of the three optimisation methods used to test the proposed system.

3.1.1. The Whale Optimization Algorithm

The WOA algorithm is a swarm-based stochastic metaheuristic algorithm inspired by the hunting behaviour of humpback whales [11]. It is based on a mathematical model updated by iterating the three search mechanisms described below:

- the "shrinking encircling prey" mechanism is exploitative and consists in moving candidate solutions (i.e. the whales) in a neighbourhood of a the current best solution in the swarm (i.e. the prey solution) by implementing the following equation:

$$\overrightarrow{x}(t+1) = \overrightarrow{x}_{\text{best}}(t) - \overrightarrow{A} * \overrightarrow{D}_{\text{best}} \quad \text{with} \quad \begin{cases} \overrightarrow{A} = 2\overrightarrow{a} * \overrightarrow{r} - \overrightarrow{a} \\ \overrightarrow{D}_{\text{best}} = 2\overrightarrow{r} * \overrightarrow{x_{\text{best}}}(t) - \overrightarrow{x}(t) \end{cases} \tag{1}$$

  where: 1) $\overrightarrow{a}$ is linearly decreased from 2 to 0 as iterations increase (to represent shrinking as explained in [7]); 2) $\overrightarrow{r}$ is a vector whose components are randomly sampled from $[0,1]$ (t is the iteration counter); 3) the "$*$" notation indicates the pairwise products between two vectors.
- the "spiral updating position" mechanism is also exploitative and mimics the swimming pattern of humpback whales towards the prey in a helix-shaped form through equations (2) and 3:

$$\overrightarrow{x}(t+1) = e^{bl} * \cos(2\pi l) * \left|\overrightarrow{d}\right| + \overrightarrow{x}_{\text{best}}(t) \tag{2}$$

  with

$$\overrightarrow{d} = \left|\overrightarrow{x}_{\text{best}}(t) - \overrightarrow{x}(t)\right| \tag{3}$$

where b is a constant value for defining the shape of logarithmic spiral; l is a random vector in $[-1, 1]$; the "$|\ldots|$" symbol indicates the absolute value of each component of the vector;

- the "search for prey" mechanism is exploratory and uses a randomly selected solution $\overrightarrow{x}_{\text{rand}}$ as "attractor" to move candidate solutions towards unexplored areas of the search space, and possibly away from local optima, according to equations (4) to (5):

$$\overrightarrow{x}(t+1) = \overrightarrow{x}_{\text{rand}}(t) - \overrightarrow{A} * \overrightarrow{D} + \text{rand} \tag{4}$$

with

$$\overrightarrow{D}_{\text{rand}} = \left| 2\overrightarrow{a} * \overrightarrow{r} * \overrightarrow{x_{\text{rand}}}(t) - \overrightarrow{x} \right|. \tag{5}$$

The reported equations implement a search mechanism which mimics movements made by whales. Mathematically, it is easier to understand that some of them refer to explorations moves across the search space, while others are exploitation move to refine solutions within their neighbourhood. To have more information on the metaphor inspiring this equations, their formulations and their role in driving the research within the algorithm framework, one can see the survey article in [6]. A derailed scheme describing the coordination logic of the three previously described search mechanism is reported in algorithm 1.

Algorithm 1 WOA pseudocode

```
Generate initial whale positions x_i, where i = 1, 2, 3, ..., NP
Compute fitness of each whale solution and identify x_best
while t < max iterations do
    for i = 1, 2, ..., NP do
        Update a, A, C, l, p
        if p < 0.5 then
            if |A| < 1 then
                Update position of current whale x_i using equation 1
            else if |A| ≥ 1 then
                x_rand ← random whale agent
                Update position of current whale x_i with equation 4
            end if
        else if p ≥ 0.5 then
            Update position of current whale x_i with equation 2
        end if
    end for
    Calculate new fitness values
    Update X_best
    t = t + 1
end while
Return x_best
```

With reference to algorithm 1, the initial swarm is generated by randomly sampling solutions in the search; the best solution is kept up to date by replacing it only when an improvement on the fitness value occurs; the optimisation process lasts for a prefixed number of iterations, here indicated with *max budget*; he probability of using the shrinking encircling rather than the spiral updating mechanism is fixed at 0.5.

3.1.2. The BAT Algorithm

The BAT algorithm is a swarm-based searching algorithm inspired from the echolocation abilities of bats [12]. Bats use sound wave emission to generate echo that measures the distance of its prey based on the loudness and time difference of the echo and sound wave. To reproduce this system and exploit it for optimisation purposes, the following perturbation strategy must be implemented:

$$f_i = f_{\min} + (f_{\max} - f_{\min}) \cdot \beta \tag{6}$$

$$v_i(t+1) = v_i(t) + (x_i(t) - x_{\text{best}}) \cdot f_i \tag{7}$$

$$x_i(t+1) = x_i(t) + v_i(t) \tag{8}$$

where x_i is the position of the candiate solution in the search space (i.e. the bat), v_i is its velocity, f_i is referred to as "wawes frequency" factor and β is a random vector in $[0,1]^n$ (where n is the dimentionality of the problem). f_{min} and f_{max} represent the lower and upper bounds of the frequency respectively. Typical values are within 0 and 100. When the bat is close to the prey (i.e. current best solution), it gradually reduces the loudness of its sound wave while increasing the pulse rate. The pseudocode depicted in algorithm 2 shows the the working mechanism of the BAT algorithm.

Algorithm 2 BAT pseudocode

Generate initial bats X_i ($i = 1, 2, 3, \ldots, NP$) and their velocity vectors v_i
Compute fitness values and find x_{best}
Initialise pulse frequency f_i at x_i
Initialise pulse rate r_i and loudness A_i
while $t <$ *max iterations* **do**
    for $i = 1, 2, 3, \ldots, NP$ **do**
        $x_{\text{new}} \leftarrow$ move x_i to a new position with equations 6–8
    end for
    for $i = 1, 2, 3, \ldots, NP$ **do**
        if rand() $> r_i$ **then**
            $x_{\text{new}} \leftarrow x_{\text{best}}$ added with a random
        end if
        if rand() $< A_i$ **and** $f(x_{\text{new}})$ improved **then**
            $X_i \leftarrow x_{\text{new}}$
            Increase r_i and decrease A_i
        end if
    end for
    Update x_{best}
    $t = t + 1$
end while
Return x_{best}

To have more detailed information on the equations used to perturb the solutions within the search space in the BAT algorithm, we suggest reading [44].

3.1.3. The Differential Evolution

The Differential Evolution (DE) algorithms are efficient metaheuristics for global optimisation based on a simple and solid framework, first introduced in [45], which only requires the tuning of three parameters, namely the scale factor $F \in [0, 2]$, the crossover ratio $CR \in [0, 1]$ and the population size NP. As shown in algorithm 3, despite using *crossover* and *mutation* operators, which are typical of evolutionary algorithms, it does not require any selection mechanism as solutions are perturbed one at a time by means of the 1-to-1 spawning mechanising from the SI field. Several DE variants can be obtained by using different combination of crossover and mutation operators [46]. The so-called "DE/best/1/bin" scheme is adopted in this study, which employs the *best* mutation strategy and the *binomial* crossover approach. Pseudocode and other details regarding these operators are available in [10].

Algorithm 3 DE pseudocode

Generate initial population x_i with $i = 1, 2, 3, \ldots, NP$
Compute fitness of each individual and identify x_{best}
while $t <$ *max iterations* **do**
  for $i = 1, 2, 3, \ldots, NP$ **do**
    $X_m \leftarrow$ mutation ▷ "best/1" as explained in [10]
    $x_{\text{off}} \leftarrow$ crossover(X_i, X_m) ▷ "bin" as explained in [10]
    Store the best individual between x_{off} and x_i in the i^{th} position of a new population
  end for
  end
  Replace the current population with the newly generated population
  Update x_{best}
end while
Return x_{best}

3.2. Datasets

Four synthetic data sets were generated using the built-in stream data generator of the "Massive Online Analysis" (MOA) software [47]. Each synthetic data set represents different data streaming scenarios with varying dimensions, clusters numbers, drift speed and frequency of concept evolution. These data sets are:

- the *5C5C* data set, which contains low dimensional data with a low rate of data changes;
- the *5C10C* data set, which contains low dimensional data with a high rate of data changes;
- the *10D5C* data set, which is a *5C5C* variant containing high dimensional data;
- the *10D10C* data set, which is a *5C10C* variant containing high dimensional data.

Moreover, the *KDD-99* data set [48], containing real network intrusion information, was also consdered in this study. It must be highlighted that the original *KDD-99* data set contains 494021 data entries representing network connections generated in military network simulations. However, only 10% of the entries were randomly selected for this study. Each data entry contains 41 features and 1 output column to distinguish the attack connection from the normal network connection. The attacks can be further classified into 22 attack types. Streams are obtained by reading each entry of the data set sequentially.

Details on the five employed data sets are given in table 1.

Table 1. Name and Description of Synthetic Datasets and Real Dataset

Name	Dimension	Clusters No.	Samples	Drift Speed	Event Frequency	Type
5D5C	5	3–5	100,000	1,000	10,000	Synthetic
5D10C	5	6–10	100,000	5,000	10,000	Synthetic
10D5C	10	3–5	100,000	1,000	10,000	Synthetic
10D10C	10	6–10	100,000	5,000	10,000	Synthetic
KDD–99	41	2–23	494,000	Not Known	Not Known	Real

3.3. Performance Metrics

To perform an informative comparative analysis three metrics were cherry-picked from the data stream analysis literature [42,43]. These are referred to as *F-Measure*, *Purity* and *Rand-Index* [49].

Mathematically, these metrics are expressed with the following equations:

$$\text{F-Measure} = \frac{1}{k}\sum_{i=1}^{k}\text{Score}_{C_i} \tag{9}$$

$$\text{Purity} = \frac{1}{k}\sum_{i=1}^{k}\text{Precision}_{C_i} \tag{10}$$

$$\text{Rand-Index} = \frac{\text{True Positive} + \text{True Negative}}{\text{All Data Instances}} \tag{11}$$

where

$$\text{Precision}_{C_i} = \frac{V_{i\text{sum}}}{n_{C_i}} \tag{12}$$

$$\text{Score}_{C_i} = 2 \cdot \frac{\text{Precision}_{C_i} \cdot \text{Recall}_{C_i}}{\text{Precision}_{C_i} + \text{Recall}_{C_i}} \tag{13}$$

$$\text{Recall}_{C_i} = \frac{V_{i\text{sum}}}{V_{i\text{total}}} \tag{14}$$

and

- C is the solution returned by the clustering algorithm (i.e. the number of clusters k);
- C_i is the the i^{th} cluster ($i = \{1, 2, \ldots, k\}$);
- V_i is the class label with the highest frequency in C_i;
- $V_{i\text{sum}}$ is the number of instances labelled with V_i in C_i;
- $V_{i\text{total}}$ is the total number of V_i instances identified in the totality of clusters returned by the algorithm.

F-Measure represents the harmonic mean of the Precision and Recall scores, where the best value of 1 indicates ideal Precision and Recall, while 0 is the worst scenario.

Purity is used to measures the homogeneity of the clusters. Maximum purity is achieved by the solution when each cluster only contains a single class.

Rand-Index computes the accuracy of the clustering solution from the actual solution, based on the ratio of correctly identified instances among all the instances.

4. The Proposed System

This article proposes "*OpStream*", an Optimised Stream Clustering Algorithm. This clustering framework consists of two main parts: the *initialisation* phase and the *online* phase.

During the initialisation phase, a number λ of data points are accumulated through a landmark time window, the unclassified points are initialised into groups of clusters via the centroid approach, i.e. generating K centroids of clusters among the points.

In the initialisation phase, the landmark time window is used to collect data points which are subsequently grouped into clusters by generating K centroid. The latter, are generated from by solving K-centroid cost optimisation problems with fast and reliable metaheuristic for optimisation. Hence, their position is optimal and lead to high-quality predictions.

Next, during the online phase, the clusters are maintained and updated using the density-based approach, whereby incoming data points with similar attributes (i.e. according to the ϵ-neighbourhood method) form dense microclusters in between two data buffers, namely *p-microclusters* and *o-microclusters*. These are converted into microclusters with CF information to store a "light" version previous scenarios in this dynamic environment.

In this light, the proposed framework is similar to advanced single-phase methods. Howevere, it requires a preliminary optimisation process to boost its classification performances.

Three variants of OpStream are tested by using the three metaheuristic optimisers described in section 3. These stochastic algorithms (as the optimisation process is stochastic) are compared against the two DenStream and CluStream state-of-the-art deterministic stream clustering algorithms.

The following sections describe each step of the OpStream algorithm.

4.1. The Initialisation Phase

This step can be formulated as a real-valued global optimisation search problem and addressed with metaheuristic of black-box optimisation. To achieve this goal, a cost function must be designed

to allow for the individualisation of the optimal position of the centroid of a cluster. These processes have to be iterated K times to then form K clusters by grouping data according to their distance from the optimal centroids.

The formulation of the cost function plays a key part. In this research, the "Cluster Fitness" (CF) function from [50] was chosen as its maximisation leads to a high intra-cluster distance, which is desirable. Its mathematical formulation, for the κ^{th} ($\kappa = 1, 2, 3, \ldots K$) cluster, is given below

$$CF_\kappa = \frac{1}{K} \sum_{\kappa=1}^{K} S_\kappa \tag{15}$$

from where it can be observed that it is computed by averaging the K clusters' Silhouettes "S_κ". These, represents the average dissimilarity of all the points in the cluster, and are calculated as follows

$$S_\kappa = \frac{1}{n_k} \sum_{i \in C_\kappa} \frac{\beta_i - \alpha_i}{\max\{\alpha_i, \beta_i\}} \tag{16}$$

where α_i and β_i are the "Inner Dissimilarity" and the "Outer Dissimilarity" respectively.

The former value measures the average dissimilarity between a data point i and other data points in its own cluster C_{κ^*}. Mathematically, this is expressed as:

$$\alpha_i = \frac{1}{(n_{k^*} - 1)} \sum_{\substack{j \in C_{\kappa^*} \\ j \neq i}} \text{dist}(i, j) \tag{17}$$

with $\text{dist}(i, j)$ being the Euclidean distance between the two points, and n_{k^*} is the total number of points in cluster C_{κ^*}. The lower the value, the better the clustering accuracy.

The latter value measures the minimum distance between a data point i to the centre of all clusters, excluding its own cluster C_{κ^*}. Mathematically, this is expressed as:

$$\beta_i = \min_{\substack{\kappa=1,\ldots,K \\ k \neq \kappa^*}} \left(\frac{1}{n_k} \sum_{\substack{j \in C_\kappa \\ k \neq \kappa^*}} \text{dist}(i, j) \right) \tag{18}$$

where n_{k^*} is the number of points in cluster C_{κ^*}. The higher the value, the better the clustering.

These two values are contained in $[-1, 1]$, whereby 1 indicates ideal case and -1 the most undesired one.

A similar observation can be done for the fitness function CF_κ [50]. Hence, the selected metaheuristics have to be set-up for a maximisation problem. This is not an issue since every real-valued problem of this kind can be easily maximised with an algorithm designed for minimisation purposes by simply timing the fitness function by -1, and vice-versa.

Regardless of the dimensionality of the problem n, which depends on the data set (as shown in table 1), all input data are normalised within $[0, 1]$. Thus, the search space for all the optimisation process is the hyper-cube defined as $[0, 1]^n$.

4.2. The Online Phase

Once the initial clusters have been generated, by optimising the cost function formulated in section 4.1, clustered data points must be converted into microclusters. This step requires the extraction of CF vectors. Subsequently, a density-based approach is used to cluster data stream online.

4.2.1. microclusters Structure

In OpStream, each CF must contain four components, i.e. CF= [N, $\overrightarrow{\text{LS}}$, $\overrightarrow{\text{SS}}$, timestamp], where

- N$\in \mathbb{N}$ is the number of data points in the microclusters;

- $\overrightarrow{\text{LS}} \in \mathbb{R}^n$ is the linear sum of the data points in the microcluster, i.e.

$$\overrightarrow{\text{LS}} = \sum_{i=1}^{N} \overrightarrow{x_i};$$

- $\overrightarrow{\text{SS}} \in \mathbb{R}^n$ is the squared sum of the data points in the microclusters i.e.

$$\overrightarrow{\text{SS}}[j] = \sum_{i=1}^{N} \left(\overrightarrow{x_i}[j]\right)^2; \qquad j = 1, 2, 3, \ldots, n$$

- timestamp indicates when the microclusters was last updated and it is needed to implement the ageing mechanism, used to remove outdated microclusters while new data accumulated in the time window are available, defined via the following equation

$$\text{age} = T - \text{timestamp} \tag{19}$$

where T is the current time-stamp in the stream a threshold, referred to as β, is used to discriminate between suitable and outdated data points.

From CF, the centre c and radius r of a microclusters are computed as follows:

$$c = \frac{\overrightarrow{\text{LS}}}{\text{N}} \tag{20}$$

$$r = \sqrt{\frac{\overrightarrow{\text{SS}}}{\text{N}} - \left(\frac{\overrightarrow{\text{LS}}}{\text{N}}\right)^2} \tag{21}$$

as indicated in [18,43].

The obtained r value is used to initialise the ϵ-neighbourhood approach (i.e. $r = \epsilon$), leading to the formation of microclusters as explained in section 2. This microclusters, which is derived from a cluster formed in the initialisation phase, is now stored in the *p-microclusters* buffer.

4.2.2. Handling Incoming Data Points

In OpStream, for each new time window, a data point p is first converted into a "degenerative" microclusters m_p containing a single point and having the following initial CF properties:

$$\begin{aligned} m_p.\text{N} &= 1 \\ m_p.\overrightarrow{\text{LS}}_i &= p_i \qquad i = 1, 2, 3 \ldots, n \\ m_p.\overrightarrow{\text{SS}}_i &= p_i^2 \qquad i = 1, 2, 3 \ldots, n \\ m_p.\text{timestamp} &= T \end{aligned}$$

Subsequently, initial microclusters have to be merged. This task can efficiently be addressed by considering pairs of microclusters, say e.g. m_i and m_j, and computing their Euclidean distance $\text{dist}(c_{m_i}, c_{m_j})$. If m_i is the cluster to be merged, its radius r must be worked out as shown in section 4.2.1 and then be merged with m_i if

$$\text{dist}(c_{m_i}, c_{m_j}) \leq \epsilon \qquad (\epsilon = r). \tag{22}$$

Two microclusters satisfying the condition expressed with equation 22 are said to be "density-reachable". The process described above is repeated until there are no longer density-reachable

microclusters. Every time two microclusters are merged, e.g. m_i and m_j, the CF properties of the newly generated microclusters, e.g. m_k, are assigned as follows:

$$\begin{aligned} m_k.\mathrm{N} &= m_i.\mathrm{N} + m_j.\mathrm{N} \\ m_k.\overrightarrow{\mathrm{LS}} &= m_i.\overrightarrow{\mathrm{LS}} + m_j.\overrightarrow{\mathrm{LS}} \\ m_k.\overrightarrow{\mathrm{SS}} &= m_i.\overrightarrow{\mathrm{SS}} + m_j.\overrightarrow{\mathrm{SS}} \\ m_k.\mathrm{timestamp} &= T \end{aligned}$$

where T is the time at which the two microclusters were merged.

When the condition in equation 22 is no longer met by a microclusters, this is moved to the *p-microclusters* buffer. If the newly added microclusters and other clusters in the *p-microclusters* buffer are density-reachable, then they are merged. Otherwise, a new independent cluster is stored in this buffer.

This mechanism is performed by a software agent, referred to as the "Incoming Data Handler" (IDH), whose pseudocode is reported in algorithm 4 to further clarify this process and allow for its implementation.

Algorithm 4 IDH Pseudocode

```
Input: Data point p
Convert p into micro cluster m_p
Initialise merged = false
for mc in p-microclusters do
    if merged is false then
        if m_p is density reachable to mc then
            if new radius ≤ ε_mc then
                Merge m_p with mc
            else
                Add m_p to p-microclusters
            end if
            merged = true
        end if
    end if
end for
if merged is false then
    for each mc in o-microclusters do
        if merged is false then
            if m_p is density-reachable to mc then
                if new radius ≤ ε_mc then
                    Merge m_p with mc
                    merged = true
                end if
            end if
        end if
    end for
end if
if merged is false then
    Add m_p to o-microclusters
end if
end
return
```

4.2.3. Detecting and Forming New Clusters

Once microclusters in the *o-microclusters* buffer are all merged, as explained in section 4.2.2, only the minimum possible number of microclusters with the highest density exist. The microclusters with the highest number of points N is then moved to an empty set C to initialise a new cluster. After calculating its centre c, with equation 20, and radius r, with equation 21, the ϵ-neighbourhood method is again used to find density-reachable microclusters. Among them, a process is undergone to detect

the so-called *border microclusters* [36] inside C, which obviously are not present during the first iteration as C initially contains only one microclusters. Border microclusters are defined as density reachable microclusters that have density level that is below the density threshold of the first microclusters present in C. Having a threshold that is too high, cluster C will not expand, whilst having a value that is too low, cluster C will contain dissimilar microclusters. Based on the experimental data from the original paper [36], 10% threshold yields good performance.

Once the border microclusters are identified, only surrounding microclusters that are density reachable to the non-border microclusters are moved to form part of C, according to the process indicated in section 4.2.2. Figure 1 graphically depicts C. The microclusters marked in red colour does not form as part of C because it is density reachable only to a border microclusters of C.

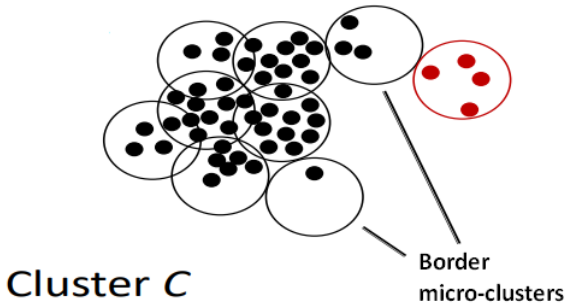

Figure 1. A graphical representation of the "border microclusters" concept [36]

This process is iterated as shown in algorithm 5. The final version of *C* is finally moved to the most appropriate buffer according to its size, i.e. if "C.N≥ minClusterSize" all its microclusters are merged together and the newly generate cluster *C* is moved to the *p-microclusters* buffer. If this does not occur, the cluster *C* is not generated by merging its microclusters but they are simply left in the *o-microclusters* buffer. The recommended method to fix the minClusterSize parameter is

$$\text{minClusterSize} = \begin{cases} 2 & \text{if 10\% of } \lambda \leq 2 \\ 10\% \text{ of } \lambda & \text{otherwise} \end{cases} \tag{23}$$

These tasks are performed by the New Cluster Generator (NCG) software agent, whose pseudocode is shown in algorithm 5.

Algorithm 5 New Cluster Generation Pseudocde

```
Input: o-microclusters
while o-microclusters is not empty do
    Initialise cluster C using mc with highest N
    addedMc = true
    while addedMc is true do
        addedMc = false
        for mc in o-microclusters do
            if mc is density-reachable to any non-border mc in C then
                Add mc into C
                Remove mc from o-microclusters
                addedMc = true
            end if
        end for
    end while
    if the size of C is ≥ minClusterSize then        ▷ minClusterSize is initialised with equation 23
        Merge microclusters mc in C
        Add C into p-microclusters
    end if
end while
```

4.3. OpStream General Scheme

The proposed OpStream method involves the use of several techniques, such as metaheuristic optimisation algorithms, density-based and k-means clustering, etc. and requires a software infrastructure coordinating activities as those performed by the IDH and NCG agents. Its architecture is outlined with the pseudocode in algorithm 6

Algorithm 6 OpStream Pseudocode

```
Launch AS                                              ▷ initialised with equation 24
initialisedFlag = false
while stream do
    Add data point p into window
    if initialisedFlag is true then
        Handle incoming data streams with IDH          ▷ i.e. algorithm 4
    end if
    if window is full then
        if initialisedFlag is false then
            Optimise centres positions and initialise clusters   ▷ e.g. with algorithm 1, 2 or 3
            initialisedFlag = true
        else
            Look for and generate new clusters with NCG          ▷ i.e. algorithm 5
        end if
    end if
end while
```

It must be added that an Ageing System (AS) is constantly run to remove outdated clusters. Despite its simplicity, its presence is crucial in dynamic environments. An integer parameter β (equal to 4 in this study) is used to compute the *age threshold* as shown below

$$age\ threshold = \beta \cdot \lambda \tag{24}$$

so that if a microclusters has not been updated in 4 consecutive windows will be removed from the respective buffer.

5. Experimental Setup

As discussed in section 3, OpStream performances are evaluated across four synthetic data sets and one real data set using three popular performance metrics. Two deterministic state-of-the-art stream clustering algorithms, i.e. DenStream [18] and CluStream [25], are also run with the suggested parameter settings available in their original articles for caparison purposes.

The WOA algorithm was initially picked to implement OpStream framework, as this framework is currently being intensively exploited for classification purposes, but two more variants employing BAT and DE (as described in section 3) are also run to 1) show the flexibility of the OpStream to the use of different optimisation methods; 2) display its robustness and superiority to deterministic approaches regardless of the optimiser used; 3) establish the preferred optimisation method over the specific data sets considered in this study. For the sake of clarity, these three variants are referred to as WOAS–OpStream, BAT–OpStream and DE–OpStream to represent its respective metaheuristic optimiser used. To reproduce the results presented in this article, the employed parameter setting of each metaheuristic, as well as other algorithmical details, are reported below:

- WOA: swarm Size = 20;
- BAT: swarm sieze = 20, $\alpha = 0.53$, $\gamma = 4.42$, $r_i = 0.42$, $A_i = 0.50$ ($i = 1, 2, 3, \ldots, n$);
- DE: population Size = 20, $F = 0.5$, $CR = 0.5$;
- the "*max Iterations*" value is set to 10 for all the three algorithms to unsure a fair comparison (the computational budget is purposely kept low due to the real-time nature of the problem);

- the three optimisation algorithms are equipped with the "toroidal" correction mechanism to handle infeasible solutions, i.e. solutions generated outside of the search space (a detailed description of this operator is available in [10]).

Furthermore, the following parameter values are also required to run the OpStream framework:

- $\lambda = 1000, \epsilon = 0.1, \beta = 4$;

Section 7 explains the role played by these parameters and how their suggested values were determined.

Thus, a total of five clustering algorithms are considered in the experimentation phase. These were executed, with the aid of the MOA platform [47], for 30 times over each data set (instances order is randomly changed for each repetition) to produce, for each evaluation metric, average $\pm$ standard deviation values. To further validate our conclusions statistically, the outcome of the Wilcoxon Rank-Sum Test [51] (with confidence level equal to 0.05) is also reported in all tables with the compact notation obtained from [52], where 1) "+" symbol next an algorithm indicated that the it is outperformed by the reference algorithm (i.e. WOA–OpStream); 2) a "−" symbol indicates that the reference algorithm is outperformed; 3) a "=" symbol shows that the two stochastic optimisation processes are statistically equivalent.

6. Results and Discussion

A table is prepared for each evaluation metric, each one displaying average value, standard deviation and the outcome of the Wilcoxon Rank-Sum test (W) over the 30 performed runs. The best performance on each data set is highlighted in boldface.

Table 2 reports the results in terms of F-measure. According to this metric, the three OpStream variants generally outperform the deterministic algorithms. The only exception is registered over the *KDDC–99* data set, where DenStream displays the best performance. From the statistical point of view, WOA–OpStream is significantly better than CluStream (with five "+" out of five cases), clearly preferable to DenStream (with four "+" out of five cases), equivalent to the DE–OpStream variant and and approximately equivalent the BAT–OpStream.

Table 2. Average F-measure value $\pm$ Standard Deviation and Wilcoxon Rank-Sum Test (reference = WOA–OpStream) for WOA–OpStream against BAT–OpStream, DE–OpStream, DenStream and CluStream on each data set.

Data set	WOA–OpStream	BAT–OpStream	W	DE–OpStream	W	DenStream	W	CluStream	W
5D5C	**0.924 ± 0.042**	0.907 ± 0.041	+	0.923 ± 0.040	=	0.645 ± 0.016	+	0.584 ± 0.033	+
5D10C	0.868 ± 0.042	0.873 ± 0.048	=	**0.879 ± 0.036**	=	0.551 ± 0.019	+	0.602 ± 0.008	+
10D5C	0.903 ± 0.031	0.899 ± 0.028	=	**0.904 ± 0.030**	=	0.619 ± 0.021	+	0.398 ± 0.006	+
10D10C	0.873 ± 0.035	**0.878 ± 0.028**	=	0.876 ± 0.027	=	0.543 ± 0.020	+	0.380 ± 0.004	+
KDDC–99	0.460 ± 0.000	0.460 ± 0.000	=	0.460 ± 0.000	=	**0.650 ± 0.000**	-	0.140 ± 0.000	+

Similarly, regarding table 3, WOA–OpStream shows a slightly better statistical behaviour than BAT–OpStream, and it is statistically equivalent to DE–OpStream, also inters of Purity. However, according to this metric, the stochastic classifiers do not outperform thee deterministic ones but have quite similar performances. In terms of average value over the 30 repetitions, DE–OpStream and DenStream have the highest purity.

Table 3. Average Purity ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = WOA–OpStream) for WOA–opStream against BAT–OpStream, DE–OpStream, DenStream and CluStream on each data set.

Data set	WOA–OpStream	BAT–OpStream	W	DE–OpStream	W	DenStream	W	CluStream	W
5D5C	0.998 ± 0.006	0.996 ± 0.007	+	0.998 ± 0.006	=	**1.000 ± 0.000**	=	0.998 ± 0.004	=
5D10C	0.992 ± 0.016	0.984 ± 0.022	=	0.987 ± 0.022	=	**1.000 ± 0.000**	-	0.998 ± 0.004	-
10D5C	**1.000 ± 0.000**	0.998 ± 0.004	+	**1.000 ± 0.000**	=	**1.000 ± 0.000**	=	**1.000 ± 0.000**	=
10D10C	0.999 ± 0.002	**1.000 ± 0.002**	=	**1.000 ± 0.002**	=	**1.000 ± 0.000**	=	**1.000 ± 0.000**	=
KDDC–99	**1.000 ± 0.000**	**1.000 ± 0.000**	=	**1.000 ± 0.000**	=	**1.000 ± 0.000**	=	0.420 ± 0.000	+

Finally, the same conclusions obtained with F-measure are drawn by interpreting the results in table 4, where the Rand-Index metric is used to evaluate classification performances. Indeed, all three OpStream variants statistically outperform the deterministic methods. This goes to show that the proposed method is performing very well regardless of the optimisation strategy, and it is always better or competitive with state-of-the-art algorithms. Unlike the case in table 2, the best performances in terms of average value or those obtained with DE rather than WOA. However, the difference between the two variants is minimal and the Wilocoxon Rank-Sum test does not detect differences between the two variants.

Table 4. Average Rand-Index ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = WOA–OpStream) for WOA–OpStream against BAT–OpStream, DE–OpStream, DenStream and CluStream on each strdata set.

Data set	WOA–OpStream	BAT–OpStream	W	DE–OpStream	W	DenStream	W	CluStream	W
5D5C	**0.951 ± 0.018**	0.945 ± 0.019	+	0.951 ± 0.017	=	0.825 ± 0.005	+	0.596 ± 0.041	+
5D10C	0.944 ± 0.020	0.947 ± 0.020	=	**0.949 ± 0.016**	=	0.753 ± 0.013	+	0.625 ± 0.018	+
10D5C	0.934 ± 0.017	0.932 ± 0.016	=	**0.935 ± 0.017**	=	0.814 ± 0.007	+	0.432 ± 0.033	+
10D10C	0.939 ± 0.020	0.941 ± 0.018	=	**0.942 ± 0.017**	=	0.746 ± 0.016	+	0.400 ± 0.020	+
KDDC–99	0.620 ± 0.000	0.620 ± 0.000	=	0.620 ± 0.000	=	0.820 ± 0.000	-	**0.940 ± 0.000**	-

Summarising, OpStream displays the best global performance, with WOA–OpStream and DE–OpStream being the most preferable variants. Statistically, WOA–OpStream and DE–OpStream have equivalent performances over different data sets and according to three different evaluation metrics. In this light, the WOA variant is preferred as requiring the tuning of only two parameters, against the three required in DE, to function optimally.

A final observation can be done by separating results from synthetic data sets and *KDDC–99*. If in the first case the supremacy of OpStream is evident, a deterioration of the performances can be noted when the later data set is used. In this light, one can understand that the proposed method presents room for improvement of handling data streams with uneven distribution of class instances as those presented in *KDDC–99* [53].

7. Further Analyses

In the light of what observed in section 6, the WOA algorithm is to be preferred over DE and BAT to perform the optimisation phase. Hence, it is reasonable to consider the WOA–OpStream variant as the default OpStream algorithm implementation.

This section concludes this piece of research with a thorough analysis of this variant in terms of sensitivity, scalability, robustness and flexibility to handle overlapping multi-density clusters.

7.1. Scalability Analysis

A scalability analysis is performed to test how OpStream behaves, in terms of execution time (seconds) needed to process 100,000 data points per data set, over data sets having increasing dimension values or increasing number of clusters. Data sets suitable for this purpose are easily generated with the MOA platform, as previously done for the comparative analysis in section 6.

This experimentation was performed in a personal computer equipped with an AMD Ryzen 5 2500U Quad-Core (2.0GHz) CPU Processor and 8GB RAM. Opstream was run with the following parameter setting: $\lambda = 1000$, $\epsilon = 0.1$, $\beta = 4$, WOA swarm size equal to 20 and maximum number of allowed iterations equal to 10.

Execution time is plotted over increasing dimension values (for the the data points) in figure 2.

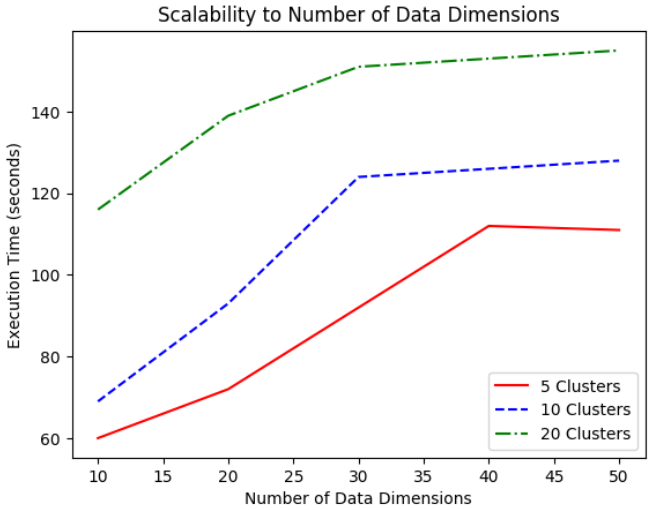

Figure 2. Scalability to Number of Data Dimensions (data dimension value).

Execution time is plotted over increasing number of clusters (in the the data sets) in figure 3.

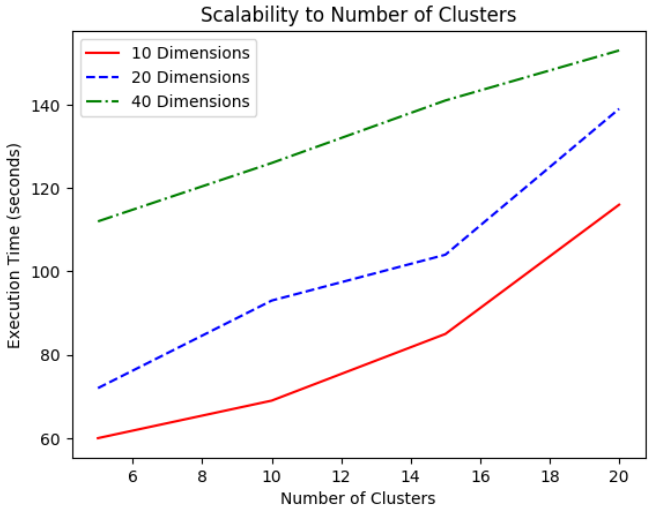

Figure 3. Scalability (number of clusters).

Regardless of the number of clusters, execution time seems to grow linearly with the dimensionality of the data points, for low dimension values, to then saturate when the dimensionality is high. Lower the number of clusters, later the saturation phenomenon takes place. With 5 clusters

this occurs at approximately 40 dimension values. In the case of 20 clusters, saturation occurs earlier at approximately 25 dimension values. This is one of the strengths of the proposed method, as its time complexity does not require polynomial times.

Conversely, no saturation takes place when execution time is measured by increasing the number of clusters. Also in this case, the time complexity seems to grow linearly with the number of clusters.

7.2. Noise Robustness Analysis

The MOA platform allows for the injection of increasing noise levels into the dataset *5D10C* and *10D5C* data sets.

The five noise levels indicated in figure 5 and 6 were used and the OpStream algorithm was run 30 times for each one of the 10 classification problems (i.e. five noise levels $\times$ 2 data sets) with the same parameter setting used in section 7.1. Results were collected to display average F-Measure, Purity and Rand-Index relative to *5D10C*, i.e. table 5, and *10D5C*, i.e. table 6.

Table 5. Average OpStream performances over or *5D10C* at multiple noise levels.

Noise Level	F-Measure	Purity	Rand-Index
0%	0.846	0.986	0.934
3%	0.798	0.988	0.909
5%	0.808	0.983	0.892
8%	0.768	0.993	0.881
10%	0.774	0.972	0.880

Table 6. Average OpStream performances over or *10D5C* at multiple noise levels.

Noise Level	F-Measure	Purity	Rand-Index
0%	0.902	1.000	0.936
3%	0.856	1.000	0.899
5%	0.854	1.000	0.889
8%	0.848	1.000	0.884
10%	0.835	1.000	0.865

From these results, it is clear that OpStream is able to retain approximately 95% of its original performance as long as the level does not exceed the 5% level. Then, performances slightly decrease OpStream seems to be robust to noise, in particular when classifying data sets with high dimensional data points and a low number of clusters number.

7.3. Sensitivity Analysis

Five parameters must be tuned before using OpStream for clustering dynamic data streams. In this section, the impact of each parameter on the classification performance is analysed in terms of Rand-Index value.

To perform a thorough sensitivity analysis

- the size λ of landmark time window model is examined in the range $[100, 5000] \in \mathbb{N}$;
- the ϵ value for the ϵ-neighbourhood method is examined within $[0, 1] \in \mathbb{R}$;
- the effect of the *age threshold* is examined by tuning β in the interval $[1, 10] \in \mathbb{N}$;
- the WOA swarm sizes under analysis are obtained by adding 5 candidate solutions per experiment, from an initial value of 5 candidate solutions to a maximum of 30 candidate solutions;

- the computational budget for the optimisation process, expressed in terms of "max iterations" number, is increased by 5 iterations per experiment starting with 5 up to a maximum of 30 iterations.

OpStream was run on three data sets for this sensitivity analysis, namely *5D10C*, *10D5C* and *KDDC–99*, and results graphically shown in the figures reported below.

Figure 4 shows that too high window sizes are not beneficial and the best performances are obtained win the rage $[500, 2000]$ data points. In particular, a peak is obtained with a size of 1000 for the two artificially prepared data sets. Conversely, slightly inferior sizes might be preferred for the *KDDC–99* data set. In general, there is no need in using more than 2000 data points are the performance will remain constant or slightly deteriorate.

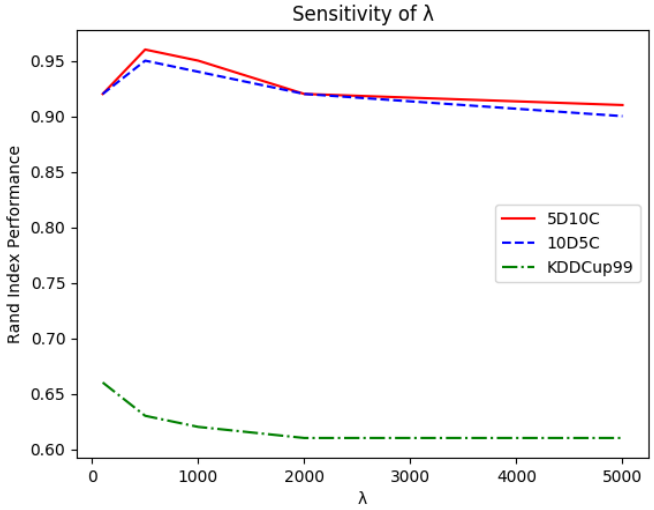

Figure 4. Sensitivity to the windows size parameter λ.

With reference to figure 5, it evident that ϵ does not require fine-tuning in a wide range as the best performances are obtained within $[0.1, 0.2]$ and then linearly decreases over the remaining admissible values. This can be easily explained as too low values would prevent microclusters from merging while too high values will force OpStream to merge dissimilar clusters. In both cases, the outcome would be a very poor classification. This observation facilitates the tuning process as it means that worth trying values for ϵ are 0.1 and 0.2 and perhaps one or two intermediary values.

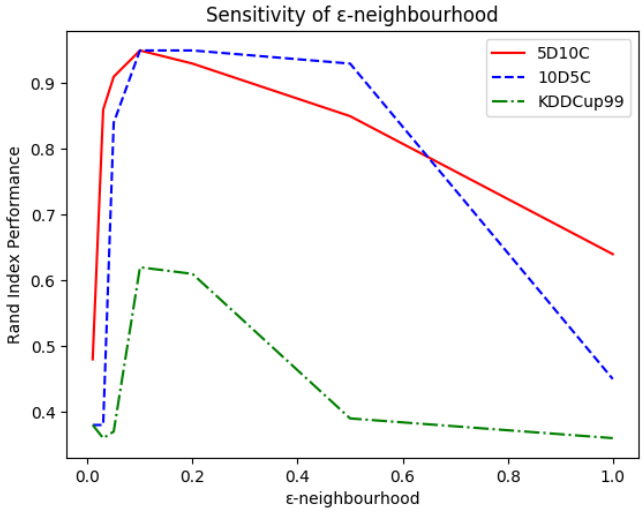

Figure 5. Sensitivity to the ϵ-neighboured parameter ϵ.

As for β, the curves in figure 6 show that OpStream is not sensitive to the value chosen for removing outdated clusters as long as $\beta \geq 2$. This means that clusters can be technically be left in the buffers for a long time without affecting the performance of the classifier. From a more practical point of view, for memory issues, it is preferable to free buffers from unnecessary microclusters timely, A sensible choice is $\beta = 4$, as too low values might prevent similar clusters from being merged due to the lack of time required for performing such process.

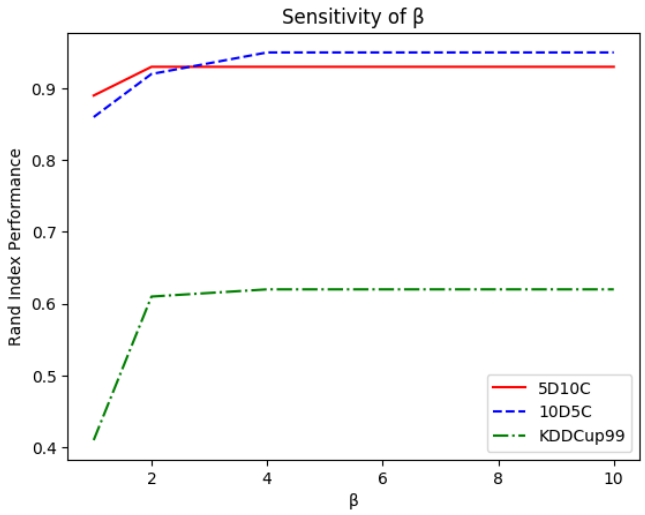

Figure 6. Sensitivity to the AS parameter β.

It can be noted that a small number of candidate solutions is used for the optimisation phase. This choice was made for multiple reasons. First, it's been recently shown that a high number of solutions can increase structural biases of the algorithm [54], which is not wanted as the algorithm has to be "general-purpose" to handle all possible scenarios obtained in the dynamic domain. Second, due to the time limitations related to the nature of this application domain, a high number of candidate solutions is to be avoided as it would slow down the converging process. This is not admissible in real-time domain where also the computational budget is kept very low. Third, as shown in figure 7 the WOA method used in OpStram seems to work efficiently regardless of the employed number of candidate solutions, as long as it is greater than 20.

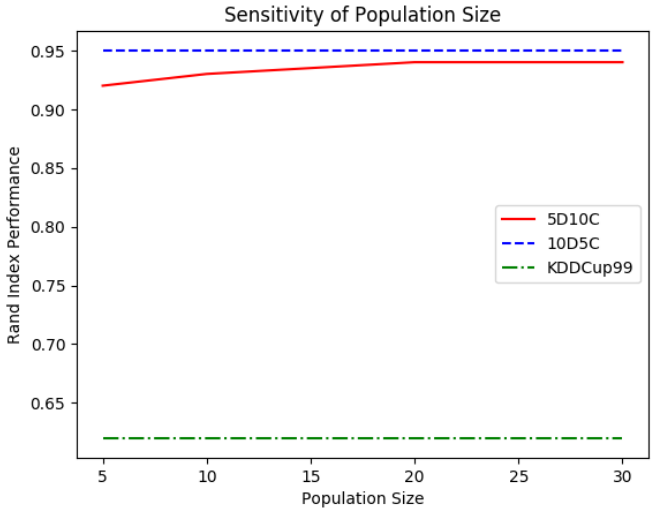

Figure 7. WOA sensitivity to the swarm size.

Similar conclusions can be done for the computational budget. According to figure 8, it is not necessary to prolong the duration of the WOA optimisation process for more than 10 iterations. This makes sense in dynamic domains where the problem changes very frequently thus making the exploitation phase less important.

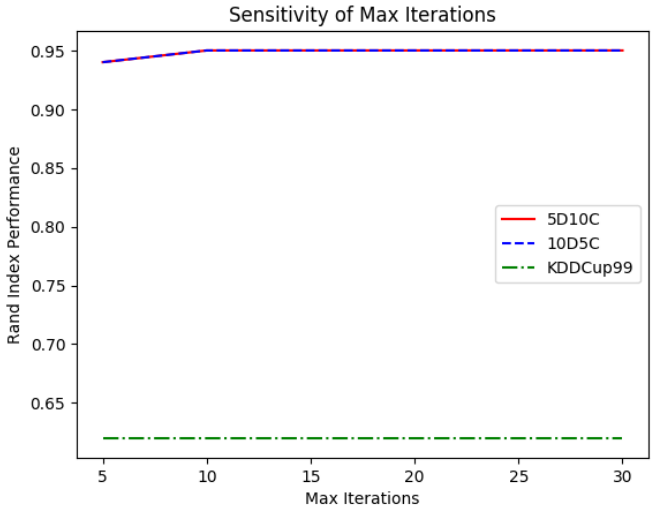

Figure 8. Sensitivity to *maxIterations*

7.4. Comparison with Past Studies on Intrusion Detection

One last comparison is performed to complete this study. This is performed on a specific application domain, i.e. network intrusion detection, by means of the "KDD–cup 99" database [55]. The comparison algorithms employed in this work, i.e. DenStream and CluStream , were both tested on this data set in their original papers [18] and [25] respectively. Despite the fact that OpStream is not meant for data sets with overlapping multi-density clusters, as in KDD–cup 99, we executed it over such data set to test its versatility. Results are displayed in table 7 where the last column indicates the average performance of the ckustering method by computing

$$\text{AVG} = \frac{\text{F-Measure} + \text{Purity} + \text{Rand-Index}}{3}. \tag{25}$$

Table 7. Results obtained with the KDD–cup 99 [55] data set for intrusion detection.

Algorithm	F-Measure	Purity	Rand-Index	AVG
OpStream	0.46	1.00	0.62	0.69
DenStream	0.65	1.00	0.82	0.82
ClusStream	0.14	0.42	0.94	0.50

Surprisingly, OpStream has an AVG better performance than CluStream, due to the fact that significantly outperforms it in terms of F-Measure and Purity, and display a state-of-the-art behaviour in terms of Purity value. As expected, DenStream provides the best performance, thus being preferable in this application domain unless a fast real-time response is required. In the latter case, its high computational cost could prevent DenStream from being successfully used [18].

8. Conclusion and Future Work

Experimental numerical results show that the proposed OpStream algorithm is a promising tool for clustering dynamic data streams as it is competitive and outperforms the state-of-the-art on

several occasions. This approach can then be applied in several challenging application domains where satisfactory results are difficult to be obtained with clustering methods. Thanks to its optimisation-driven initialisation phase, OpStream displays high accuracy, robustness to noise in the data set and versatility. In particular, we found out that its WOA implementation is efficient, scalable (both in term of data set dimensionality and number of clusters) and resilient to parameters variations. Moreover, due to a low number of parameters to be tuned in WOA, this optimisation algorithm is preferred over other approaches returning similar accuracy values as DE and BAT. Finally, this study clearly shows that hybrid clustering methods are promising and more suitable that classic approaches to address challenging scenarios.

Possible improvements can be done to address some of the aspects arose during the experimental section. First, the deterioration of the performance over unevenly distributed data sets, as *KDDC-99*, will be investigated. A simple solution to this problem is to embed non-density-based clustering algorithms into the OpStream framework. Second, since the proposed methods do not benefit from prologues optimisation processes (as shown in figure 8), probably because of the dynamic nature of the problem, optimisation algorithm employing "restart" mechanisms will be implemented and tested. These algorithms usually work on a very short computational budget and handle dynamic domains better than other by simply re-sampling the initial point where a local search routine is applied, as e.g. [56], or by also adding to it information from previously past solution with the "inheritance" method [57–59].

It is also worthwhile to extend OpStream to handle overlapping multi-density clusters in dynamic data streams, as these cases are not currently addressable and are common in some real-world scenarios, such as network intrusion detection [53] and Landsat satellite image discovery [60].

Author Contributions: All authors contributed to the draft of the manuscript, read and approved the final manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Modi, K.; Dayma, R. Review on fraud detection methods in credit card transactions. 2017 International Conference on Intelligent Computing and Control (I2C2), 2017, pp. 1–5. doi:10.1109/I2C2.2017.8321781.
2. Moodley, R.; Chiclana, F.; Caraffini, F.; Carter, J. Application of uninorms to market basket analysis. *International Journal of Intelligent Systems* **2019**, *34*, 39–49. doi:10.1002/int.22039.
3. Moodley, R.; Chiclana, F.; Caraffini, F.; Carter, J. A product-centric data mining algorithm for targeted promotions. *Journal of Retailing and Consumer Services* **2019**, p. 101940. doi:https://doi.org/10.1016/j.jretconser.2019.101940.
4. Zarpelão, B.B.; Miani, R.S.; Kawakani, C.T.; de Alvarenga, S.C. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications* **2017**, *84*, 25 – 37. doi:https://doi.org/10.1016/j.jnca.2017.02.009.
5. Masud, M.M.; Chen, Q.; Khan, L.; Aggarwal, C.; Gao, J.; Han, J.; Thuraisingham, B. Addressing Concept-Evolution in Concept-Drifting Data Streams. 2010 IEEE International Conference on Data Mining, 2010, pp. 929–934. doi:10.1109/ICDM.2010.160.
6. Gharehchopogh, F.S.; Gholizadeh, H. A comprehensive survey: Whale Optimization Algorithm and its applications. *Swarm and Evolutionary Computation* **2019**, *48*, 1 – 24. doi:https://doi.org/10.1016/j.swevo.2019.03.004.
7. Hardi M. Mohammed, S.U.U.; Rashid, T.A. A Systematic and Meta-Analysis Survey of Whale Optimization Algorithm. *Computational Intelligence and Neuroscience* **2019**, *2019*, 25. doi:https://doi.org/10.1155/2019/8718571.

8. Storn, R.; Price, K. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* **1997**, *11*, 341–359. doi:10.1023/A:1008202821328.
9. Caraffini, F.; Kononova, A.V. Structural bias in differential evolution: A preliminary study. *AIP Conference Proceedings* **2019**, *2070*, 020005, [https://aip.scitation.org/doi/pdf/10.1063/1.5089972]. doi:10.1063/1.5089972.
10. Caraffini, F.; Kononova, A.V.; Corne, D. Infeasibility and structural bias in Differential Evolution. *Information Sciences* **2019**, pp. 161 – 179. doi:https://doi.org/10.1016/j.ins.2019.05.019.
11. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Advances in Engineering Software* **2016**, *95*, 51 – 67. doi:https://doi.org/10.1016/j.advengsoft.2016.01.008.
12. Yang, X.S. A New Metaheuristic Bat-Inspired Algorithm. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)* **2010**, *284*, 65–74. doi:10.1007/978-3-642-12538-6_6.
13. Chen, G.; Luo, W.; Zhu, T. Evolutionary clustering with differential evolution. *2014 IEEE Congress on Evolutionary Computation (CEC)* **2014**, pp. 1382–1389. doi:10.1109/CEC.2014.6900488.
14. Carnein, M.; Trautmann, H. evoStream – Evolutionary Stream Clustering Utilizing Idle Times. *Big Data Research* **2018**, *14*, 101 – 111. doi:https://doi.org/10.1016/j.bdr.2018.05.005.
15. Nasiri, J.; Khiyabani, F. A Whale Optimization Algorithm (WOA) approach for Clustering. *Cogent Mathematics & Statistics* **2018**, *5*. doi:10.1080/25742558.2018.1483565.
16. Nandy, S.; Sarkar, P. Chapter 8 - Bat algorithm–based automatic clustering method and its application in image processing. *Bio-Inspired Computation and Applications in Image Processing* **2016**, pp. 157 – 185. doi:https://doi.org/10.1016/B978-0-12-804536-7.00008-9.
17. Kokate, U.; Deshpande, A.; Mahalle, P.; Patil, P. Data Stream Clustering Techniques, Applications, and Models: Comparative Analysis and Discussion. *Big Data and Cognitive Computing* **2018**, *2*, 32. doi:10.3390/bdcc2040032.
18. Cao, F.; Ester, M.; Qian, W.; Zhou, A. Density-Based Clustering over an Evolving Data Stream with Noise. *In 2006 SIAM Conference on Data Mining* **2006**, *2006*, 328–339. doi:10.1137/1.9781611972764.29.
19. Sun, J.; Fujita, H.; Chen, P.; Li, H. Dynamic financial distress prediction with concept drift based on time weighting combined with Adaboost support vector machine ensemble. *Knowledge-Based Systems* **2017**, *120*, 4 – 14. doi:https://doi.org/10.1016/j.knosys.2016.12.019.
20. Brzezinski, D.; Stefanowski, J. Prequential AUC: properties of the area under the ROC curve for data streams with concept drift. *Knowledge and Information Systems* **2017**, *52*, 531–562. doi:10.1007/s10115-017-1022-8.
21. ZareMoodi, P.; Kamali Siahroudi, S.; Beigy, H. Concept-evolution detection in non-stationary data streams: a fuzzy clustering approach. *Knowledge and Information Systems* **2019**, *60*, 1329–1352. doi:10.1007/s10115-018-1266-y.
22. Carnein, M.; Trautmann, H. Optimizing Data Stream Representation: An Extensive Survey on Stream Clustering Algorithms. *Business & Information Systems Engineering: The International Journal of WIRTSCHAFTSINFORMATIK* **2019**, *61*, 277–297. doi:10.1007/s12599-019-00576-5.
23. Gao, X.; Ferrara, E.; Qiu, J. Parallel clustering of high-dimensional social media data streams. 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2015, pp. 323–332.
24. Gao, L.; Jiang, Z.y.; Min, F. First-Arrival Travel Times Picking through Sliding Windows and Fuzzy C-Means. *Mathematics* **2019**, *7*, 221. doi:10.3390/math7030221.
25. Aggarwal, C.C.; Yu, P.S.; Han, J.; Wang, J. - A Framework for Clustering Evolving Data Streams. *Proceedings 2003 VLDB Conference* **2003**, *29*, 81 – 92. doi:https://doi.org/10.1016/B978-012722442-8/50016-1.
26. Madhulatha, T.S. Overview of streaming-data algorithms. *CoRR* **2012**, *abs/1203.2000*, [1203.2000].
27. Kokate, U.; Deshpande, A.; Mahalle, P.; Patil, P. Data Stream Clustering Techniques, Applications, and Models: Comparative Analysis and Discussion. *Big Data and Cognitive Computing* **2018**, *2*, 32. doi:10.3390/bdcc2040032.
28. Hartigan, J.A.; Wong, M.A. Algorithm AS 136: A K-Means Clustering Algorithm. *Applied Statistics* **1979**, *28*, 100–108. doi:10.2307/2346830.
29. O'Callaghan, L.; Mishra, N.; Meyerson, A.; Guha, S.; Motwani, R. Streaming-data algorithms for high-quality clustering. *Proceedings 18th International Conference on Data Engineering* **2002**, pp. 685–694. doi:10.1109/ICDE.2002.994785.

30. Spinosa, E.J.; de Leon F. de Carvalho, A.P.; Gama, J.a. OLINDDA: A Cluster-based Approach for Detecting Novelty and Concept Drift in Data Streams. *Proceedings of the 2007 ACM Symposium on Applied Computing* **2007**, pp. 448–452. doi:10.1145/1244002.1244107.
31. Forestiero, A.; Pizzuti, C.; Spezzano, G. A single pass algorithm for clustering evolving data streams based on swarm intelligence. *Data Mining and Knowledge Discovery* **2013**, *26*, 1–26. doi:10.1007/s10618-011-0242-x.
32. Forestiero, A.; Pizzuti, C.; Spezzano, G. FlockStream: A Bio-Inspired Algorithm for Clustering Evolving Data Streams. *2009 21st IEEE International Conference on Tools with Artificial Intelligence* **2009**, pp. 1–8. doi:10.1109/ICTAI.2009.60.
33. Alswaitti, M.; Albughdadi, M.; Isa, N.A.M. Density-based particle swarm optimization algorithm for data clustering. *Expert Systems with Applications* **2018**, *91*, 170 – 186. doi:https://doi.org/10.1016/j.eswa.2017.08.050.
34. Shamshirband, S.; Hadipoor, M.; Baghban, A.; Mosavi, A.; Bukor, J.; Varkonyi-Koczy, A.R. Developing ANFIS-PSO Model to Predict Mercury Emissions in Combustion Flue Gases. *Mathematics* **2019**. doi:10.3390/math7100965.
35. Kong, F.; Jiang, J.; Huang, Y. An Adaptive Multi-Swarm Competition Particle Swarm Optimizer for Large-Scale Optimization. *Mathematics* **2019**, *7*, 521. doi:10.3390/math7060521.
36. Fahy, C.; Yang, S. Finding and Tracking Multi-Density Clusters in Online Dynamic Data Streams. *IEEE Transactions on Big Data* **2019**, pp. 1–1. doi:10.1109/TBDATA.2019.2922969.
37. Dorigo, M.; Di Caro, G. Ant colony optimization: a new meta-heuristic. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), 1999, Vol. 2, pp. 1470–1477 Vol. 2. doi:10.1109/CEC.1999.782657.
38. Doan Quang Tu and A. S. M. Kayes and Wenny Rahayu and Kinh Nguyen. ISDI: A New Window-Based Framework for Integrating IoT Streaming Data from Multiple Sources. Advanced Information Networking and Applications - Proceedings of the 33rd International Conference on Advanced Information Networking and Applications, AINA 2019, Matsue, Japan, March 27-29, 2019, 2019, pp. 498–511. doi:10.1007/978-3-030-15032-7_42.
39. Krempl, G.; Žliobaite, I.; Brzeziński, D.; Hüllermeier, E.; Last, M.; Lemaire, V.; Noack, T.; Shaker, A.; Sievi, S.; Spiliopoulou, M.; Stefanowski, J. Open Challenges for Data Stream Mining Research. *SIGKDD Explor. Newsl.* **2014**, *16*, 1–10. doi:10.1145/2674026.2674028.
40. Yin, C.; Xia, L.; Wang, J. Data Stream Clustering Algorithm Based on Bucket Density for Intrusion Detection. Advances in Computer Science and Ubiquitous Computing; Park, J.J.; Loia, V.; Yi, G.; Sung, Y., Eds.; Springer Singapore: Singapore, 2018; pp. 846–850. doi:10.1007/978-981-10-7605-3_134.
41. Huang, G.; Zhang, Y.; Cao, J.; Steyn, M.; Taraporewalla, K. Online mining abnormal period patterns from multiple medical sensor data streams. *World Wide Web* **2014**, *17*, 569–587. doi:10.1007/s11280-013-0203-y.
42. Fahy, C.; Yang, S.; Gongora, M. Finding Multi-Density Clusters in non-stationary data streams using an Ant Colony with adaptive parameters. *2017 IEEE Congress on Evolutionary Computation (CEC)* **2017**, pp. 673–680. doi:10.1109/CEC.2017.7969375.
43. Fahy, C.; Yang, S.; Gongora, M. Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams. *IEEE Transactions on Cybernetics* **2019**, *49*, 2215–2228. doi:10.1109/TCYB.2018.2822552.
44. Yang, Xin-She and Hossein Gandomi, Amir. Bat algorithm: a novel approach for global engineering optimization. *Engineering Computations* **2012**, *29*, 464–483.
45. Storn, R.; Price, K. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, ICSI, 1995.
46. Opara, K.R.; Arabas, J. Differential Evolution: A survey of theoretical analyses. *Swarm and Evolutionary Computation* **2019**, *44*, 546 – 558. doi:https://doi.org/10.1016/j.swevo.2018.06.010.
47. Bifet, A.; Holmes, G.; Kirkby, R.; Pfahringer, B. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* **2010**, *11*, 1601–1604.
48. University of California, I. KDD Cup 1999, 2007.
49. Rand, W.M. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association* **1971**, *66*, 846–850.
50. Hedar, A.R.; Ibrahim, A.M.M.; Abdel-Hakim, A.E.; Sewisy, A.A. K-Means Cloning: Adaptive Spherical K-Means Clustering. *Algorithms* **2018**, *11*. doi:10.3390/a11100151.

51. Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin* **1945**, *1*, 80–83.
52. Caraffini, F. The Stochastic Optimisation Software (SOS) platform. https://doi.org/10.5281/zenodo.3237024, 2019. doi:10.5281/zenodo.3237024.
53. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications* **2009**, pp. 1–6. doi:10.1109/CISDA.2009.5356528.
54. Kononova, A.V.; Corne, D.W.; Wilde, P.D.; Shneer, V.; Caraffini, F. Structural bias in population-based algorithms. *Information Sciences* **2015**, *298*, 468–490. doi:https://doi.org/10.1016/j.ins.2014.11.035.
55. Rosset, Saharon and Inger, Aron. KDD-cup 99: Knowledge Discovery in a Charitable Organization's Donor Database. *SIGKDD Explor. Newsl.* **2000**, *1*, 85–90. doi:10.1145/846183.846204.
56. Caraffini, F.; Neri, F.; Gongora, M.; Passow, B. Re-sampling Search: A Seriously Simple Memetic Approach with a High Performance. IEEE Symposium Series on Computational Intelligence, Workshop on Memetic Computing, 2013, pp. 52–59. doi:10.1109/MC.2013.6608207.
57. Iacca, G.; Caraffini, F. Compact Optimization Algorithms with Re-Sampled Inheritance. Applications of Evolutionary Computation; Kaufmann, P.; Castillo, P.A., Eds.; Springer International Publishing: Cham, 2019; pp. 523–534. doi:10.1007/978-3-030-16692-2_35.
58. Caraffini, F.; Iacca, G.; Yaman, A. Improving (1+1) covariance matrix adaptation evolution strategy: A simple yet efficient approach. *AIP Conference Proceedings* **2019**, *2070*, 020004, [https://aip.scitation.org/doi/pdf/10.1063/1.5089971]. doi:10.1063/1.5089971.
59. Caraffini, F.; Neri, F.; Epitropakis, M. HyperSPAM: A study on hyper-heuristic coordination strategies in the continuous domain. *Information Sciences* **2019**, *477*, 186 – 202. doi:https://doi.org/10.1016/j.ins.2018.10.033.
60. Li, X.; Ye, Y.; Li, M.J.; Ng, M.K. On cluster tree for nested and multi-density data clustering. *Pattern Recognition* **2010**, *43*, 3130 – 3143. doi:https://doi.org/10.1016/j.patcog.2010.03.020.